Journal of Network and Systems Management, Vol. 5, No. 1, 1997

Towards A Role-Based Framework for Distributed Systems Management

Emil C. Lupu[1,2] and Morris Sloman[1]

Roles have been widely used for modeling the authority, responsibility, functions, and interactions, associated with manager positions within organizations. In this paper, we discuss the issues related to specifying roles for both human and automated managers of distributed computer systems. The starting point is that a role can be defined in terms of the authorization and obligation policies, for a particular manager position, which specify what actions the manager is permitted or is obliged to do on a set of target objects. This permits individuals to be assigned or removed from positions without respecifying the policies for the role. However these policies are insufficient for fully specifying relationships between managers and the targets they manage or between different manager roles. There is a need to specify the interaction protocols and how managers coordinate and synchronize their activities. The role-based framework consists of a set of tools enabling the creation of roles from policies, the specification of the concurrency constraints for role activities and the specification of protocols for role interaction. In addition, the issues related to conflicts which can occur between policies within a role or between interacting roles are briefly discussed.

KEY WORDS: Management roles; role interactions; management policy; obligation; authorization; policy conflicts.

1. INTRODUCTION

Role Theory, which is widely used for enterprise modeling, postulates that individuals occupy positions in an organization [1, 2]. Associated with each position is a set of activities including required interactions that constitute the responsibilities of that position. A role thus identifies the authority, obligations, functions and interactions, associated with a position such as vice president, board director, security administrator or operator responsible for reactor number three.

[1]Department of Computing, Imperial College, 180 Queen's Gate, London, SW7 2BZ, United Kingdom. E-mail: e.c.lupu@doc.ic.ac.uk, m.sloman@doc.ic.ac.uk

[2]Correspondence should be directed to E. C. Lupu.

1064-7570/97/0300-0005$12.50/0

Networks and distributed systems can be very large and complex requiring the partitioning of management responsibility according to management functions such as security configuration, fault or performance management, but the partitioning may also reflect organizational, geographical or even network layer boundaries. There are thus likely to be many different roles associated with the management of such systems. Role theory provides us with very useful concepts for a management framework and has been applied to other areas of Information Technology such as organizational analysis and office automation.

The development of new IT techniques has introduced the ideas of assisting human managers in their tasks. The efforts have been oriented in two directions: assisting the manager at the information level and providing support for office automation. Three approaches can be distinguished: Computer Supported Cooperative Work (CSCW) attempts to provide tools for office assistance [3–6], the object-oriented approach models the knowledge and characteristics of different roles [7–9] and the process oriented approach is directed towards office automation [10–12].

In large systems it is essential to distribute management tasks where required and if this distribution follows the organizational structure the management system can be integrated within the current functioning of the organizations. Clear specifications of the management policies, representation of the organizational structure and partial automation of the management tasks are needed in order to realize and integrate a management framework. Our initial motivation for implementing roles in the management system was to simplify the specification of policies so that they do not have to be changed when managers are assigned to new positions. The idea of "intelligent" automated agents which can be assigned to a particular role has been used in Distributed Artificial Intelligence and is being applied in areas such as information location for the World-Wide Web. The people assigned to roles do not work in isolation but interact and cooperate with other roles. Building on concepts from role theory and organizational analysis, we intend to provide a framework where both human and automated agents coexist and which provides support for the communication and coordination between these agents. The overall motivation for implementing roles in a computer system, which supports people is thus to simplify the specification of the system and to provide a framework which can model the organizational structure. Although we are aiming this at the Network and Distributed Systems Management, we consider that our framework and tool support will be useful for office automation and role-based access control as well [13].

The starting point for this work has been to define a *manager role as the set of authorization and obligation policies for which a particular manager position is the subject* [14]; so in Section 2 we explain the concepts and notation used for specifying management policies for the actions of both human and automated agents. Section 2 also describes domains as a means of grouping objects to which

a policy applies. In Section 3, we elaborate on the concept of a role as a set of policies and indicate why this is insufficient for a role framework. In Section 4, we discuss how role interaction can be specified; and in Section 5, we indicate how concurrency issues such as parallelism and synchronization can be specified. In Section 6, we present issues relating to conflicts of policies within and between roles. This is followed by an example of how the framework could be used to specify roles.

2. DOMAINS AND POLICIES

Large distributed systems may contain millions of objects so it is impractical to specify policies for individual objects. Objects are grouped in domains for specifying a common management policy or to structure and partition management responsibility. A **domain** is a collection of objects (actually references to object interfaces) which have been explicitly grouped together for the purposes of management (cf., file system directories or folders). If a domain holds a reference to an object, the object is said to be a **direct member** of that domain. A domain is an object, so may also be a member of another domain and is said to be a **subdomain**. Its members are **indirect members** of the parent domain. A **domain service** is provided for the manipulation of the membership information.Further, **domain scope expressions** can be specified determining the set of objects to which a policy applies. For example D_1-D_2-O_3 represents the objects that are members of D_1 with members of D_2 and object O_3 excluded. Our concept of a domain is very similar to that of a directory in a typical hierarchical file system. The policy which applies to a domain will, by default, propagate to subdomains and to the objects within them, although this propagation can optionally be disabled. A **User Representation Domain (URD)** is a persistent representation of the human within the computing system. When a person logs in, an adapter object (cf., login shell) is created within the URD to act as the interface process between the person and the computer system. Other agents representing the human could also be created in the URD. Details on the domain structure and the relevant services can be found in Refs. [14, 15].

Policies establish a relationship between manager and managed object domains. Domains can however exist independently of policies, they are a method of grouping objects in a large system for management purposes. A policy expresses either an **authorization**—what activities the managers are permitted or forbidden to perform or an **obligation**—what activities the managers must or must not perform on the managed objects. The **mode** of the policy distinguishes between positive authorization (permitted: A+), negative authorization (forbidden: A−), positive obligation (must: O+) and negative obligation (must not: O−). The general format of the policies is given here with optional arguments within brackets:

```
policy_id mode [ trigger ] subject { action } target
                [ when constraint ] ;
```

The subject represents the set of managers assigned to carry out the actions on the set of target objects. Both sets are specified using domain scope expressions. Obligation policies can be triggered by time or by composite events detected within the monitoring system [16]. Constraints limit the applicability of the policy, e.g., between the hours of 09.00 and 17.00. The policy format and use is described in Ref. [17]. Examples of policies are:

```
payment_1 O+ at [31/Dec/_] accountant {
    pay(percentage_of_profits, credit_transfer, pounds_sterling)
} subcontractor ;

/* On 31 of December of every year the accountant is obliged to
pay a percentage of the profits in pounds by credit transfer to
the subcontractor.  */

purchase_1 A- u:users { purchase() } services
     when u.type == anonymous ;

/* Anonymous users are forbidden to purchase services. */
```

Policies can specify actions at different levels of abstraction. A refinement hierarchy can therefore be built from the more abstract policies, which can only be interpreted by humans, to the enactable leaf level policies or rules which can be interpreted by automated components. Tools for policy editing and services have been implemented and are described in Ref. [17].

3. CONCEPTS FOR ROLE-BASED MANAGEMENT

In this section, we elaborate on how the concepts of domains and policies can be applied to modeling roles. In Ref. [1] a **Role** is defined as "*a collection of rights and duties*" and a **Position** describes a status within the organization. The role specifies management actions (i.e. policies) which represent the behavior or dynamic aspects of the position which is essentially a static concept. We loosen this definition to consider a role to be composed of a **Manager Position Domain** and a dynamic part consisting of a set of authorization and obligation policies

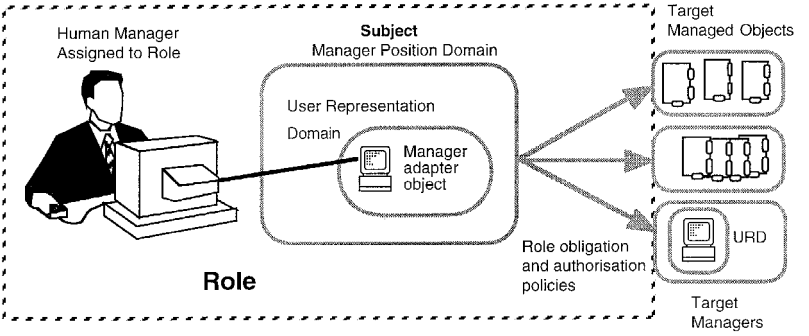

Fig. 1. Position domains and roles.

which have the position domain as a subject. Assigning a manager to a position merely implies including the manager's URD into the position domain. Role policies will propagate to the URD and apply to the adapter object or other agents in the URD (see Fig. 1). Multiple URDs may be included in a position domain to represent sharing of a position by a number of managers and a URD can be included in several position domains if the manager performs multiple roles [14]. The advantage of using a position domain as the subject of the policies is that individuals (or automated agents) can be assigned to or withdrawn from a role without having to respecify policies. In addition a position can temporarily exist without appointed managers to represent a vacant role. Note that the role policies can specify activities relating to target managed objects or other manager positions indicating a relationship such as supervision.

A manager must be able to delegate his duties and/or his access rights to another provided he is authorized to do so. The need for delegation occurs in various situations such as subcontracting between organizations, when a manager is on leave and someone else takes on his roles or a manager assigns a role to another manager. A very simple form of **delegation** of a role from Manager A to Manager B can be achieved by replacing A's URD in the position domain by B's URD.

In general polices propagate to subdomains of a parent domain which is a subject or target of a policy. Propagated policies may apply to a position domain but are not a component of the role associated with that position. For example the College policy for the use of computers propagates to all the research students but is not a specific part of the obligations and authorizations of their research role. A manager having his URD included in other organizational domains (research groups, experts on a particular subject) will be subject to other policies than those specified within the role(s) he is assigned to.

It would be very useful to be able to define a role template which can then

be parameterized with specific target domains. It would thus be possible to define the role policy set as a class from which particular instances can be created. For example a role class could be defined for a region manager and this could be used to create North, South, East, and West region roles. Each of the four roles instances has its own manager position and specific target domains, but specifies the same policy activities and constraints for each role. An advantage of an object-oriented approach to specifying roles and policies classes is that inheritance can be used to define new role classes in terms of the existing ones [18]. Policy classes (used for defining role classes) are templates with variable subject or target domains. For example many companies have standard templates for their contractual agreements with subcontractors. The template specifies standard terms and conditions for obligations of the company and contractor without knowing which of the company agents will implement them and to which subcontractors they apply.

It is possible to identify many different relationships between manager positions within an organization. There is typically a hierarchical relationship which reflects organizational structure: e.g., President, Vice-President, regional managers, site managers within a company. These hierarchical relationships include issues of delegation of responsibility from a superior to a junior manager, supervision of activities of a junior and a junior manager reporting events or status to a more senior manager. A second form of relationship occurs as part of the tasks performed by managers [19]. For example the design and production managers have to negotiate a start date for production of the new components and the designs have to be completed before production can start. These task relationships include resource sharing, information access and coordination. Some roles provide a service to others and so client-server and contractual relationships have to be considered. The role framework has to be very flexible to accommodate peer-to-peer as well as hierarchical relationships and to permit multi-party interactions. A committee with chairman, secretary and cooperating members is a good example of mixing all these types of role relationships.

Relationships cannot be fully specified by policies as there is a need to define **interaction protocols** in terms of message content and permitted message sequences; e.g., for contract negotiation (see Section 4). Multiple managers may be assigned to a role and concurrently perform the role policy activities so there is a need to synchronize completion or ordering of activities within a role. Policies can use events disseminated by the monitoring system to synchronize their activities, but this is a low-level and difficult means of specifying coordination and synchronization between roles so there is a need for a higher level notation for the specification of **concurrency constraints** (see Section 5). For other studies relating to inter role relationships and possibilities for their graphical representation see Refs. [20–23].

As represented in Fig. 2 our framework identifies for a role position: (i)

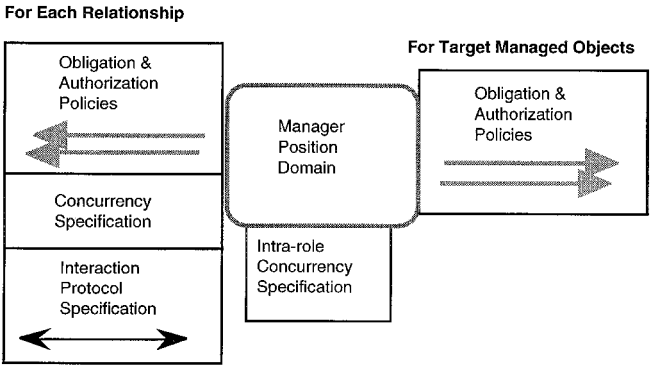

Fig. 2. Components of an Extended Role.

the authorization and obligation policies related to target objects which may be other managers or shared resources, (ii) the relationships between roles which reflect the organizational structure, and (iii) both intra- and inter-role concurrency constraints.

4. ROLE INTERACTION

Managers performing different roles have to interact in order to achieve their tasks, coordinate their actions, share information and agree on their commitments. This section attempts to elaborate on the issues of interaction protocols between roles and proposes a framework suitable for both human and automated managers. The interaction protocol is related to the role activity and is a part of the specification of a relationship. When using the term *role interaction* we refer to interactions between manager objects within role position domains.

4.1. Related Work

Communication protocols in computer networks and telecommunication disciplines have often been specified using finite state machines or Petri-Nets. This approach is largely unsatisfactory when dealing with complex interactions among human agents—the main limitations of these notations being:

- All the possible 'states' of the agents in the interaction have to be foreseen.
- Two party interactions can be specified but the notation introduces undesirable complexity for multi-party interactions.
- The messages exchanged have no semantic value—they are transitions from one state to another.

A different approach, by computer scientists from an organizational and artificial intelligence background, attributes a meaning; i.e., a semantic value to the type of exchanged messages and derives knowledge on the behavior of the manager from the message. This meaning is termed an "illocutionary" value. For example a statement such as "The URL of our document archive is `ftp://dse.doc.ic.ac.uk/dse-papers`" has an assertive value on the state of the system while a statement such as "I will perform the backup of the system by 9.00 pm" has a commisive value on the activities of the manager. The first system introducing such ideas was **The Coordinator** [24, 25]. Winograd identifies the concept of Conversation for Action (CfA) based on the Speech Act theory [26]. Higher level protocol specifications follow this line, among them logics such as Modal Action Logic [27], Conversational Clichés [28] and Information System design [29]. This work is largely inspired by the desire to support cooperation between agents representing human managers.

De Greef's notation [11] inherits features of a logic programming language, provides sets and introduces the notion of agents. The notation is powerful enough to express multi-agent algorithms such as the Contract Net [30]. Support for multi-agent interaction is provided by the use of sets of receiver agents and by the ability to collect messages from multiple senders. The interaction specification is the description of a network style graphical specification also used in Winograd's work. This notation has been implemented in April [31].

An approach based on Petri-Nets has been taken by Singh [32], where interactions refer to a bi-directional synchronous mechanism for the coordination of agents. Multi-party interactions can be specified constraining all the interacting agents to synchronize. A graphic representation can be used for interaction specification with the horizontal dimension representing the different interacting roles and the vertical dimension representing time. Singh's specifications have a direct translation into Petri-Nets.

4.2. Issues for Role Interaction

The issues relating to a role interaction protocol are identified and discussed in the following subsections.

4.2.1. Human vs. Automated Interactive Entities

A communication protocol between automated agents has to be deterministic. Although heuristic approaches can be developed which include choices to represent non-determinism in the protocol, the behavior of the automated agent then becomes unpredictable. Interactions occurring between human managers are always partly non-deterministic due to human choice. For example, a human manager might respond to a counter proposal by a request for further details even

if this was not specified in the initial protocol. Most of the work in the area constrains the human managers to the pre-defined protocol and concentrates on capturing the illocutionary value of the messages exchanged. The protocol we are aiming at should provide a flexible framework for communication between the human agents as well as a deterministic approach for communication between automated managers. Determinism for automated managers can be ensured by restricting the use of the choice operators.

4.2.2. Two-Party vs. Multi-Party Interaction

The problem of multi-party interactions is an underlying concern when modeling interactions among agents. Although techniques such as the Contract Net have been available for a long time, most of the recent work does not deal with multi-party interactions. We believe that multi-party interactions are essential; e.g., for specifying joint decisions or task allocation for roles. De Greef's notation [11], and the April [31] language identify the need for sets in the interaction specification language to express the multiple senders or receivers in an interaction and for collecting the multiple messages received by one manager.

4.2.3. The Illocutionary Value of the Messages Exchanged

In order to automatically derive a meaning from the messages exchanged they must be typed (request, reply, proposal, statement, etc.) and a finite Universe of Discourse (UoD) [29] must define the set of all the types of messages used within the interaction. An illocutionary value can then be given to each type of message and rules can be built for deriving knowledge of the state of the system and the activities of the managers. We do not discuss illocutionary value in this paper but make provision for it and recognize its benefits for assisting the managers in their tasks.

4.2.4. The Content of the Exchanged Messages

In the early studies of the interaction protocols the value of the message is the value of the illocutionary act it represents. These are entirely predefined in the protocol and a message has the same value regardless of the stage the interaction protocol has reached. Schmidt and Simone describe the case of a "bug form" which is filled in by the various managers in the process of the development of a software product [33]. The success of this protocol relies partially on the fact that the constraints placed upon the different managers and the past interactions between them are explicitly represented within the exchanged message—the bug form. In particular, when the designer receives the bug form he has knowledge of the different other stages—tests, specification scheme, etc. This highlights one of the main limitations of the actual protocol specification. In both The Coordinator and the de Greef specification language, a received message triggers the

same action regardless of the previous "history" of the exchanged messages.[2] Specifications like "release patch only after receiving two testing reports" are desirable and should be accommodated.

4.2.5. Interaction as a Means of Specifying Synchronization

This approach is largely adopted in Ref. [32] where all the different parts involved in multi-party interactions have to synchronize. Although interactions can specify part of the synchronization between managers, the synchronization is distributed in the role or manager specifications among the different interaction protocols and is difficult to determine or analyze. In particular, synchronization specification, based on interaction, cannot include events relating to the completion of a task. For example, using messages, it is difficult to specify that a manager has to report to the sales director when a customer contract is signed and must then arrange delivery of materials to the relevant subcontractors and arrange for the manufactured components to be delivered and assembled before delivery to the customer. This requires interactions based on common events or messages in the manager, sales manager, customer, and subcontractors. A high-level notation which includes events for beginning/end of activities is desired. Concurrency specification is detailed in Section 5.

4.3. The Proposed Interaction Specification Notation

The specification notation is separated into a global specification of the sequence of messages and a local production rule-based specification for each manager, in order to achieve the flexibility for an interaction system that includes human managers. The global specification includes the description of the interacting parts and of a pre-defined sequence of messages. It will be graphical, but can be automatically transformed into a set of production rules which can then be changed, if necessary, within a manager.

4.3.1. Configuring a Sequence of Messages

The global specification is aimed at giving a high-level means of specifying the minimal sequence of messages exchanged in an interaction between the related parties. It therefore contains the message types exchanged by the managers and offers a global view of the interaction. Configuring a sequence of messages to define permitted message ordering can be achieved with the network style specification used by both de Greef and Winograd. This also has the advantage of representing the interaction as a finite state machine with each state representing a manager at some point in the interaction [11, 25]. It is therefore

[2] In April [31] (an implementation of the de Greef specification language) variables can be maintained locally in order to remember the previously exchanged messages. This approach is not satisfactory since it implies keeping logs of the messages exchanged with each manager.

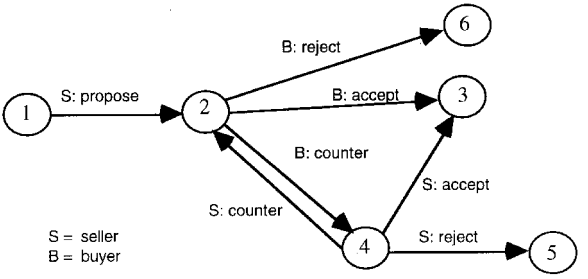

Fig. 3. The haggling protocol.

possible to translate this global specification into sets of production rules by representing the various regular expressions recognizing each state of the finite state machine. The regular expressions match a sequence representing the history of the exchanged messages (present in the incoming message) and trigger production rules. The production rules are distributed to each manager and constitute the local specification of the protocol. This representation by regular expressions is possible because the messages are typed and the Universe of Discourse is finite. The example in Fig. 3 shows the sequence of messages for the haggling over a price between a buyer and a seller (from Ref. [11]).

This specification is minimal but represents the messages that are essential for the interaction to proceed and the initial/final states of the protocol.

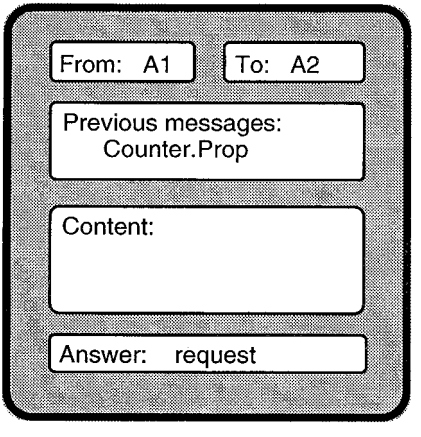

Fig. 4. A message form.

4.3.2. A Local Specification for the Messages Exchanged and the Production Rules

4.3.2.1. The content of a message. The exchanged message should contain the sequence of the types of the previous exchanges in the interaction (we assume here trusted interactions). This is like the bug form which carries the signature of the different agents finding, analyzing or correcting the bug. Including the type (but not the content) of the previous exchanges in the message allows the receiving manager to have knowledge about the previous steps of the interaction. The message should also contain the source manager and the destination managers. A special feature that will enable changes in the protocol is to allow the sender of the message to specify additional types of desired answers. Figure 4 shows a typical "message form" representation.

In textual form a message is represented by:

```
message: 'TO:' <destination>, 'FROM:' <sender>,
         'LOG:' /* sequence of the types of the
                   messages exchanged during the interaction */
         'CONTENT:' <content>
         'ANSWER': <possible answer>
```

As an example the message

```
[TO: accountant, FROM: Mr. Beecham, LOG: request.payment,
     CONTENT: card_no: xxx exp_date: xxx, ANSWER: more_id]
```

means that the accountant has received a payment from Mr. Beecham subsequent to a request for payment. The payment was made by a credit card and Mr. Beecham offers the accountant the possibility to reply by requesting additional identification elements. 'request', 'payment' and 'more_id' must belong to the Universe of Discourse and the sequence `request.payment.more_id` can match a regular expression and trigger one of the rules defined in the protocol of the accountant.

4.3.2.2. Formal specification. A production rule is composed of a regular expression, a guard and a block of instructions giving the different reply possibilities. The regular expressions match the sequence of types of the messages exchanged (`Log` field of the incoming message) and trigger the rules. Guards are predicates expressed on the content of the message. The guards provide the means to distinguish between messages of the same type and with

the same history in order to trigger different rules. The body of the production rule consists of a sequence of activities to be performed. Sending a message (including replying to the incoming message) is considered as an activity. Different alternatives are separated by the keyword '**or**'. This keyword gives a freedom of choice for the human manager but its use is restricted for automated managers. Alternative answers can also be expressed within the message of specifying the type of the answer accepted in the 'answer' field of the message.

It should be easy to include the possibility of deriving commitments or abstracting plans in a system based on production rules triggered by regular expressions. Interactions based on abstraction of plans are investigated in feature interaction within intelligent telecommunication systems [34, 35].

4.3.2.3. Production rules for the protocol specification. The possible replies a human manager can send to a message are given by the set of production rules local to the manager. In the following example, we represent the sending of a message by a tuple [type, content, additional answer] -> destination. The production rules for the buyer and seller in the haggling protocol shown in Fig. 3 are as follows.

Seller

```
(1) [propose, price, ] -> buyer

(3) propose.{counter}.counter =>
      [reject, reasons, ] -> buyer
       or {[accept, , ] - > buyer ; order_delivery }
       or [counter, new_price, ] -> buyer,

(4) *.accept => order_delivery;

(5) *.reject =>
```

Rule (1) specifies that the seller should initiate the interaction by proposing a price. Rule (3) specifies that after receiving the initial proposal and one or more counter-offers, the seller can reject the last counter-offer by giving reasons, can accept it and order the delivery of the product or can make a new counter-offer. Rule (4) indicates that the seller must order delivery if the proposal is accepted and rule (5) indicates no action if the proposal is rejected.

Buyer

```
(2) propose.{counter} => [reject, reasons, ] -> seller
 or [accept, desired_delivery_date, ] -> seller
 or [counter, new_offer, ] -> seller

(6) *.reject =>

(7) *.accept => payment;
```

Rule (2) specifies that after receiving a proposal and zero or more counter offers, the buyer can reject giving reasons, accept with a desired delivery date or return a counter offer with a new price.

The advantage of such a notation is that new rules can be very easily added to change the alternatives defined by the protocol. Also a rule can be changed without needing to recompile the whole protocol specification again. For example, rule (8) could be added for the seller to permit rejection of an offer only after at least two counter offers (assuming reject is not present in rule (3):

```
(8) propose.counter2.{counter} => [reject, reasons, ] -> buyer
```

4.3.2.4. Dynamic aspects. In order to indicate the dynamic features that can be used in this framework we assume that the seller offers the possibility to the buyer to request additional information about the offered item. The seller needs to indicate an extension to the universe of discourse which is accomplished by a counter message with **moreinfo** as a parameter in the **answer** field; i.e., the seller now can now respond to a **moreinfo** message. Rule (9) is derived from rule (3) to achieve this.

```
(9) propose.{counter}.counter =>
            [counter, newask, moreinfo] -> buyer
            or [reject, reasons, ] -> buyer
            or [accept, delivery, ] -> buyer
```

The rules for the buyer would need to be extended to permit the generation of a **moreinfo** request, but this is not shown.

4.3.2.5. Discussion. Including humans along with automated managers in

a management framework introduces trade-offs between the flexibility and assistance desired by a human manager and the efficiency aimed at in automation. This is particularly true in the case of interaction protocols. Classical network interaction protocol specifications such as ESTEREL [36] assume a deterministic specification and concentrate on the efficiency of the interaction. The protocol specification is compiled and can be used to generate outline code for an application. It is assumed that the protocol specification remains unchanged during the lifetime of the applications. The various alternative answers to a message are all known when the protocol is specified and the activities are all performed within the application. Specifying protocols which can change in ESTEREL would mean foreseeing and making provision for all the possible permitted answers (answer field of the message) and thus expressing choices which may possibly cover all the Universe of Discourse. Introducing human managers in the set of interacting agents implies making provision for the human choice although this limits part of the protocol to human implementation. In order to adapt the protocol to the manager's requirements the specification must be flexible enough to allow changes without complete recompiling, as is possible with the rule based interpreted approach. Finally in order to assist the human manager, knowledge should be derived from the exchanged messages or from the history of the protocol; i.e., the set of previous messages within the interaction.

5. CONCURRENCY OF ACTIVITIES WITHIN POLICIES

As mentioned previously there is a need to specify the parallelism and synchronization between the activities within roles and between roles. There are many well known notations for specifying concurrency among processes—among them CCS, CSP, and Petri-Nets are widely used. Our aim is not to provide a new calculus system for the specification of concurrency but rather to provide an easy way of specifying sequences of activities within the role framework. In this section, we propose a graphical notation based on Petri-Nets which can then be translated into a textual notation.

5.1. Graphical Concurrency Notation

An activity expression is represented by square box preceded and followed by a transition (circle) representing an event (token). The required concurrency operators are represented in Fig. 5.

This Petri Net style representation offers the advantage of representing the events as tokens triggering the different activities, but this does increase the complexity of the representation. Although Petri Nets style representations are convenient for human usage they need to be translated into a textual notation for computer based interpretation.

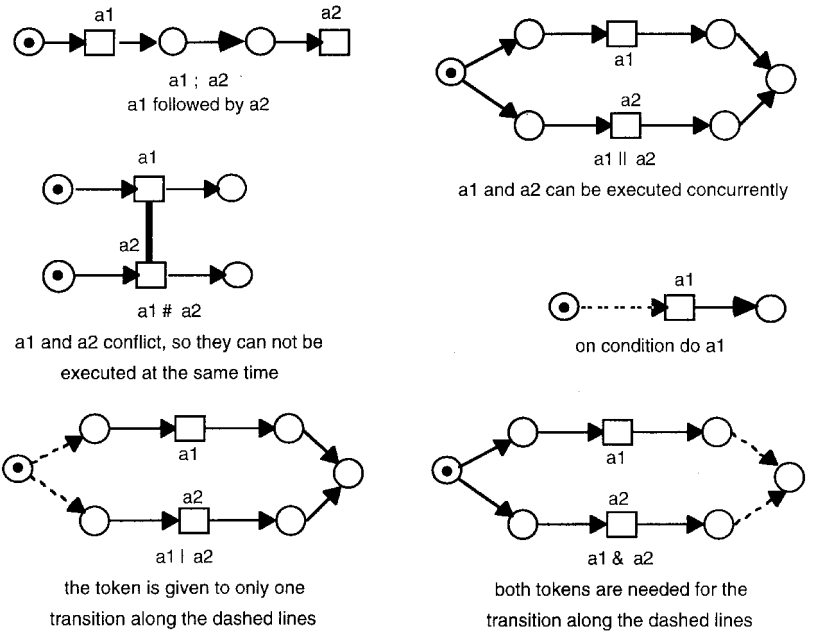

Fig. 5. Petri Net style currency representation.

5.2. Textual Concurrency Notation

A role consists of policies which specify activities. We refer to the activity a2 within the policy P3 of the role "Accountant" by `Accountant:P3:a2`. A sequence of activities separated by semicolons forms a block enclosed in brackets:

```
{accountant:p2:register_payment; accountant:p3:issue_check}.
```

An activity can be followed by a sequence:

```
R1:p2:act2 {R2:p3:act4 ; R1:p3:act5}.
```

However the specification of a sequence of activities must also provide ways of expressing concurrency. We introduce the "||" operator for this purpose. `A1 || A2` allows the two activities to be executed concurrently and the execution of the sequence continues when any of these activities are completed. Alternatives between activities can be expressed with the "|" operator:

```
{{accountant:p2:register_payment;accountant:p3:issue_check} |
accountant:p4:delay_payment} accountant:p5:produce_report
```

The "&" operator expresses the need for synchronized activities, e.g., `A1 & A2` states that both A1 and A2 can be executed concurrently but they both have to be completed before the sequence proceeds. We also permit conditional expressions of the form:

{on (condition) do sequence}

Finally an operator "#" is needed to indicate conflicting activities which must not be executed concurrently; e.g., `account:p2:update_records# accountant:p3: backup_records.`

5.3. Event Based Semantics

Policies are triggered by events monitored within the distributed system and we use a notation called GEM [16] to combine simple events to form composite, more abstract events. The semantics of the concurrency notation can be defined in terms of events for the beginning and end of an activity execution. Events can also be used to express an order between two sequences of activities. The notation of an activity in a sequence is therefore extended to permit optional events; e.g., [event] activity [event] specified by the user. All events, other than those explicitly declared by the user, will be automatically generated by a translator from the concurrency specification. The concurrency operators can be translated into compound event declarations. In the following examples, `ix` is an event indicating start of activity `ax` and `ix f` is an event indicating the end of activity `ax`.

```
a1i1f ; i2a2 => i2 = i1f
    /* The event i1f marking the end of a1 is assigned to i2 for
    the start of a2. If a1 is explicitly triggered by an event
    e2 then i2 >= i1 (i2 must occur after i1) and
    i2 = i1f & e2*/

i1f{i2 a2 i2f & e3 a3 i3f} i4 =>
      i2 = i1f, i3 = i1f & e3, i4 = i2f & i3f
    /* The event i2 triggering a2 is given by i1f. The event
    triggering a3 is i3 = i1f & e3 (e3 is declared by the user).
    The event finishing the sequence is i4 = i2f & i3f. */
```

This semantics based on events is consistent with the other tools used in this framework. The events used in the monitoring system are also used to trigger policies and the receipt of a message during an interaction also generates an event. It is therefore possible to inter-relate the policy triggering mechanism with the interactions and the specifications of sequences of activities. For example an activity a1 can be performed only if another activity a2 has finished and an event message is received allowing a1 to continue.

6. CONFLICTS

In this section, we briefly discuss the conflicts which may occur between the policies and the possible conflict detection mechanisms. Policies reference managers and managed objects as sets within domains. Preventing these sets from overlapping is not practical nor desirable. A classification of the conflicts based on the overlapping sets of managers, activities and managed objects is presented in Ref. [37]. We will discuss here two types of conflicts, namely modality and application specific conflicts.

6.1. Types of Conflicts

Modality conflicts occur if there is both a positive and negative authorization or obligation policy with overlap of subjects, targets and activities (see Fig. 6). Work in this area, restricted to the authorization policies, can be found in Ref. [38]. Our conflict detection mechanism distinguishes between **potential conflicts** which are detected by analyzing the overlaps between the domain scope expressions and **real conflicts** needing precedence relationships based on domain nesting or priorities in order to resolve the conflict. There is a need to be able to include objects in multiple domains and to specify multiple policies for a domain so it is not possible to prevent conflicts. The aim of the conflict detection mechanism is then to find all the real conflicts among the potential ones. There

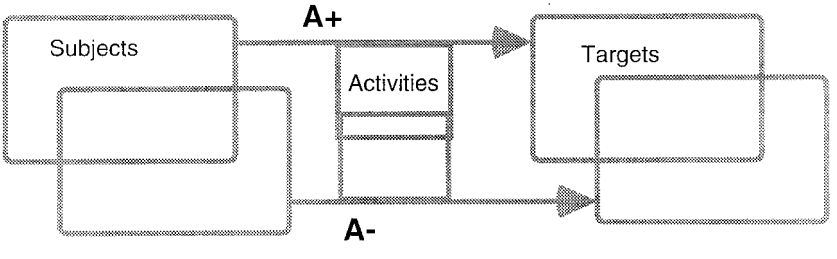

Fig. 6. Conflict of authorization.

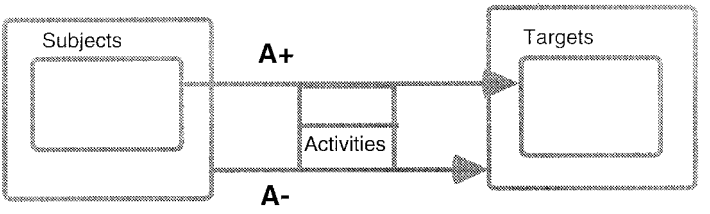

Fig. 7. Domain nesting.

are various criteria for establishing the precedence between conflicting policies, among them priorities, distance [39], and nesting [38]. We have experimented with domain nesting to establish both an idea of distance between the policy and the objects to which it is applied, and of priorities. In Fig. 7 the positive authorization takes precedence over the negative since it refers to "inner" domains; e.g., the system administrator is authorized to change the system files although users generally are forbidden.

Application specific conflicts cannot be determined by examining overlaps or policy modalities. For example the same manager is not permitted to both enter a payment and sign the check for the payment. We specify these conflicts using **meta policies** (i.e., policies about policies) in the form of a logical predicate over the managers, activities and managed objects sets expressed in the policies. The predicate is a normal form containing membership operations on these sets. This type of expression can be written or translated in a logic based programming language such as Prolog which we use to specify the meta policies to detect the conflicts from [37]. For example, the above conflict can be specified as:

```
any(P1,P2) belonging (policy_scope_expression)
      false <- intersect (P1.managers, P2.managers)
            && belongs('enter', P1.activities)
            && belongs('sign', P2.activities)
            && belongs('check', P2.targets)
            && belongs('payment', P1.targets)
```

6.2. The Scope of Conflict Detection

Conflict checking must ensure that the related parts do not have conflicting obligations and that there is a corresponding authorization policy permitting the actions specified within an obligation policy. Role definition and implementation cannot be centralized in a large system, so the cost of conflict checking in terms

of computation can be very high. A single role is centralized and so can be checked comparatively easily. This will ensure that all the activities assigned to the managers sharing the position can be performed. The explicit specification of relationships between roles identifies the distributed roles which must also be checked for inter-role conflicts and so provides some bounds on the problems of scale. Tool support for conflict detection is described in Ref. [42].

7. EXAMPLE: MULTIMEDIA PURCHASING FRANCHISE

We consider the relationships between the customers (users), a software retailer (Presentation_AG) and its subcontractors. The Presentation_AG allows customers to purchase products from subcontractors in a multimedia session. The users can purchase the products and pay the retailer for the products purchased. The retailer has a sales manager responsible for the sale of the subcontractor products to the users and a security manager responsible for the authentication of users.

This example outlines the relationship between the user role and the security manager role (Fig. 8). This relationship governs their behavior when interacting with one another for establishing a connection between the users and the Presentation_AG agent. Policies relating to each role's obligations and authorizations describe their responsibilities while the interaction protocol specifies the order of the exchanged messages. The policies are:

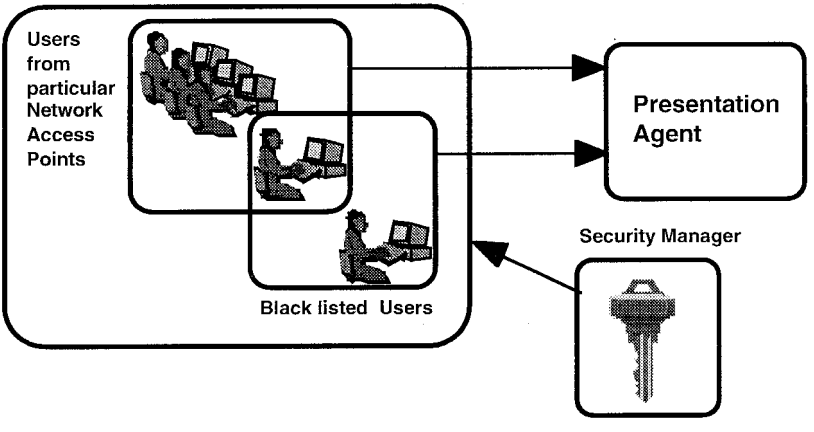

Fig. 8. Connection to the presentation agent.

```
p_access_1 A+ users/NAP[3]/England + users/PA[4]/England
      { connect_to() } Presentation_AG ;

p_access_2 A- u:BlackList_users { connect_to() } Presentation_AG ;

p_access_3 O+ security_manager {
      /* permanent address should not be used to deny access */
      } u:Users
      when u.type == corporate ;

p_access_4 A+ security_manager { suspend() } user_connection;

p_access_5 O+ on connection_request security_manager
      {/* authenticate */} u:users ;

p_access_6 O+ on fraudulent_connection_attempt security_manager {
      enter(action) } security_log ;
```

These policies specify that users accessing the service from England and users with a permanent address in England are authorized to connect. Users on the Black List are not authorized to connect. These two policies can create a conflict if a user belongs to both the domains **users/NAP/England** and **BlackList_users**. This modality conflicts can be detected by domain overlap and automatically solved by giving precedence to the denial of access. The policies also specify that the security manager is authorized to suspend a connection and any fraudulent connection attempt will be logged. The security manager has to authenticate the user and corporate users should not be denied access when located at their permanent address. The interaction protocol is given by the following production rules:

User

```
[connect, ,]  -> security manager

connect.request :: request = authentication =>
       [send, (client_no, PIN), ] -> security manager
/* if the request regards the authentication the user sends its
client number and its PIN */
```

[3]Network Access Point
[4]Permanent Address

Security Manager

```
connect  => [request, (client_no, PIN) , ] -> user
connect.request.id :: content = <client_no, PIN> &&
     (member(client_no,corporate users) ||
        member(From, authorized NAP) ||
        client_no = anonymous)
=>  valid = check(client_no, PIN);
        if (valid) {
              [authorization, authorization_Id , ] -> user
               or [denial , reasons, ] -> user
              /* the security manager can still deny
              access for  other reasons */
        } else {
              event(fraudulent_connection_attempt);
              enter(action, security_log);
        }

/*  The  restrictions  on  the  access  are  implemented  within
the  interaction protocol. The guard implements the criteria
on which the authorization to connect is issued.*/
```

This example also outlines the connection between the policies and the interaction protocols. Policy p_access_5 is implemented by the interaction protocol since it specifies the authentication of the user. The receipt of a message constitutes an event which can trigger a policy—the security manager has to authenticate the user on receipt of a connection request.

Finally the sequence of activities in the related roles is given by:

```
user:request_connection ; security_manager:authenticate_user ;
[fraud_connection_attempt] security_manager:enter(action,log)
```

This example also outlines the need to include a concurrency specification within the formal description of the relationship.

8. CONCLUSIONS AND FURTHER WORK

Tools for specifying policies and maintaining the refinement hierarchy have been implemented [17] together with the domain service to support this. We have experimented with implementing conflict detection for a set of policies although further work is required on this. We are working on tools to support the proposed notation for specifying interactions and concurrency and integrate this within an object oriented role specification framework [42].

There is increasing interest in role based access control and our concepts of roles and authorization policy can be used for specifying this. Our framework could also be used for organizational analysis to explicitly specify and analyze roles within organizations. There is a need for this sort of framework for managing intelligent networks—for example a virtual private network would have three overlapping domains corresponding to the network provider, service provider and customer [40, 41] with at least one (and probably more) manager roles relating to each domain.

The key concepts identified in our role framework are a role as a set of policies relating to a particular manager position and the notation to specify policies. This was extended to include role interactions and concurrency as part of the inter-role relationship specification. The framework must cater for a variety of role relationships. In particular, the formalism defined has to offer the flexibility required for human managers and good support for office automation. Tools are also needed to analyze the policies within and between related roles for conflicts and inconsistencies.

One of the problems we have identified arises when a human manager is assigned to multiple roles so the URD representing the manager is included in multiple domains. All the role policies propagate to the manager and it is not possible for the manager to have sessions in which some of the roles are enabled and others are disabled. We are investigating a solution in which the URD is not included in the position domain, but instead a manager agent is included in the position domain and a session is set up between the object adapter in the manager's URD and the agent in the position domain. It would then make it easier for the manager's object adapter to distinguish between sessions corresponding to particular roles; e.g., by mapping a session to a particular window on the manager's terminal.

ACKNOWLEDGMENTS

We acknowledge financial support from the EPSRC RoleMan project (No. GR/K37512). We are grateful to BT for permission to use the Travel Co scenario as an example of management policies and support on the Management of Multiservice Networks Project. Many thanks for comments which

have improved both the content and the form of this report to René Wies, Nicholas Yialelis, Nikolaos Skarmeas, Damian Marriott, and Masoud Mansouri-Samani.

REFERENCES

1. B. J. Biddle and E. J. Thomas, *Role Theory: Concepts and Research*, Robert E. Krieger Publishing Company, New York, 1979.
2. B. J. Biddle, *Role Theory, Expectations Identities and Behaviour*, Academic Press Inc., 1979.
3. K. Crowston, T. W. Malone, and F. Lin, Cognitive science and organisational design: a case study of computer conferencing. In I. Greif, (ed.) *Computer Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann Publishers Inc., pp. 713–740, 1988.
4. T. W. Malone, K.-Y. Lai, and C. Fry, Experiments with oval: a radically tailorable tool for cooperative work, *ACM Transactions on Information Systems*, Vol. 13, No. 2, pp. 177–205, 1995.
5. T. W. Malone, Computer support for organisations: towards an organisational science, MIT Sloan School of Management, Working Paper 85-012, Management in the 1990s, September 1985.
6. C. A. Ellis and G. J. Nutt, Office information systems and computer science, In I. Greif, (ed.) *Computer Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann Publishers Inc., pp. 199–247, 1988.
7. B. Pernici, Objects with Roles, *ACM Conference on Office Information Systems*, Cambridge Massachusetts, pp. 205–215, 1990.
8. B. Singh, Interconnected Roles (IR): A Coordination Model, MCC, Technical Report CT-084-92, July 1992.
9. C. Martens and F. H. Lochovsky, OASIS: A programming environment for implementing distributed organizational support systems, *ACM Conference of Organizational Computing*, Atlanta (Georgia-U.S.), pp. 29–42, 1991.
10. N. Skarmeas, Organisations through roles and agents, *International Workshop on the Design of Cooperative Systems (COOP'95)*, Antibes-France, 1995.
11. P. de Greef, K. L. Clark, and F. G. McCabe, Towards a specification language for cooperation methods, *16th German AI-Conference, GWAI'92*, Berlin, pp. 313–320, 1992.
12. C. Bußler, Capability based modelling, *First International Conference on Enterprise Integration Modelling*, pp. 389–398, 1992.
13. E. C. Lupu, D. A. Marriott, M. S. Sloman, and N. Yialelis, A policy-based role framework for access control, *First ACM/NIST Role-Based Access Control Workshop*, Gaithersburg, USA, December 1995.
14. M. S. Sloman, Policy driven management for distributed systems, *Journal of Network and Systems Management*, Vol. 2, No. 4, pp. 333–360, 1994.
15. K. P. Twidle, Domain services for distributed systems management, Imperial College, Department of Computing, Ph.D. Thesis 1993.
16. M. Mansouri-Samani and M. S. Sloman, GEM-A generalised event monitoring language for distributed systems, *IEE/IOP/BCS Distributed Systems Engineering Journal*, Vol. 4, No. 2, June 1997.
17. D. A. Marriott and M. S. Sloman, Management policy service for distributed systems, *IEEE Workshop on Services in Distributed and Networked Environments*, Macau, June 1996, pp. 2–9.
18. R. S. Sandhu, E. J. Coyne, H. L. Feinstein, and C. E. Youman, *IEEE Computer*, Vol. 29, No. 2, pp. 38–47, 1996.
19. R. A. Hirscheim, Understanding the office: a social-analytic perspective, *ACM Transactions on Office Information Systems*, Vol. 4, No. 4, pp. 331–344, 1986.

20. T. W. Malone and K. Crowston, The interdisciplinary study of coordination, *ACM Computing Surveys*, Vol. 26, No. 1, pp. 87–119, 1994.
21. O. A. Oeser and F. Harary, A mathematical model for structural role theory, II *Human Relations*, No. 17, pp. 3–17, 1964.
22. O. A. Oeser and F. Harary, Role Structures: A description in Terms of Graph Theory. In B. J. Biddle, (ed.), *Role Theory: Concepts and Research*, Robert E. Krieger Publishing Company, New York, pp. 92–102, 1979.
23. Z. Milosevic, A. Berry, A. Bond, and K. Raymond, Supporting business contracts in open distributed systems, *Second International Workshop on Services in Distributed and Networked Environments*, Whistler, British Columbia, California, pp. 60–67, 1995.
24. F. Flores, M. Graves, B. Hartfield, and T. Winograd, Computer systems and the Design of Organisational Interaction, *ACM Transactions on Office Information Systems*, Vol. 6, No. 2, pp. 153–172, 1988.
25. T. Winograd, A language/action perspective on the design of cooperative work. In I. Greif (ed.), *Computer Supported Cooperative Work: A Book of Readings*, Morgan Kaufmann Publishers Inc., pp. 623–653, 1988.
26. J. R. Searle, *Speech Acts: An Essay* in the *Philosophy of Language*, Cambridge University Press, 1969.
27. J. Pitt, M. Anderton, and J. Cunningham, Normalized interaction between autonomous agents: a case study in Interorganizational Project Management, *International Workshop on the Design of Cooperative Systems (COOP'95)*, Antibes, France, pp. 76–95, 1995.
28. C. Laufer and H. Fuks, ACCORD conversation cliches for cooperation, *International Workshop on the Design of Cooperative Systems (COOP'95)*, Antibes, France, pp. 351–369, 1995.
29. P. Johannesson, Representation and communication—a speech act based approach to information systems design, *Information Systems*, Vol. 20, No. 4, pp. 291–303, 1995.
30. R. G. Smith, The contract net protocol: high-level communication and control in a distributed problem solver, *IEEE Transactions on Computers*, Vol. 29, No. 12, pp. 1104–1113, 1980.
31. F. G. McCabe and K. L. Clark, April—Agent process interaction language. In N. Jennings and M. Wolldridge, *Intelligent Agents*, Vol. 890, *Lecture Notes in Artificial Intelligence*, Springer-Verlag, 1995.
32. B. Singh and G. L. Rein, Role interaction nets (RINs): A process description formalism, MCC, Technical Report CT-083-92, July 1992.
33. K. Schmidt and C. Simone, Mechanisms of interaction: an approach to CSCW systems design, *International Workshop on the Design of Cooperative Systems (COOP'95)*, Antibes, France, pp. 56–75, 1995.
34. N. D. Griffeth and H. Velthuijsen, Reasoning about goals to resolve conflicts, *Int. Conf. on Intelligent Cooperative Information Systems*, Los Alamitos, California, pp. 197–204, 1993.
35. H. Velthuijsen, Distributed artificial intelligence for runtime feature-interaction resolution, *IEEE Computer*, pp. 48–55, August 1993.
36. G. Berry and G. Gonthier, The Esterel synchronous programming language: design, semantics, implementation, *Journal of Science of Computer Programming*, Vol. 19, No. 2, pp. 87–152, 1992.
37. J. D. Moffett and M. S. Sloman, Policy conflict analysis in distributed system management, *Ablex Publishing Journal of Organisational Computing*, Vol. 4, No. 1, pp. 1–22, 1994.
38. A. Heydon, M. Maimone, J. Tygar, J. Wing, and A. Zaremski, Miro: visual specification of security, *IEEE Transactions on Software Engineering*, Vol. 16, No. 10, pp. 1185–1197, 1990.
39. M. M. Larrondo-Petrie, E. Gudes, H. Song, and E. B. Fernandez, Security policies in object-oriented databases, *IFIP Database Security: Status and Prospects*, pp. 257–268, 1990.
40. B. Alpers and H. Plansky, Concepts and application of policy-based management. In A. S. Sethi,

Y. Raynaud, and F. Faure-Vincent, (Eds.), *Integrated Network Management IV*, Santa-Barbara California Chapman and Hall, pp. 57–68, 1995.
41. B. Alpers, H. Plansky, and R. Sauerwein, Applying domain and policy concepts to customer network management, *International Switching Symposium*, 1995.
42. E. C. Lupu and M. S. Sloman, Conflict analysis for management policies, *IFIP/IEEE International Symposium on Integrated Network Management*, San-Diego, California, May 1997.

Emil C. Lupu is currently a research assistant at Imperial College, London, working towards the Ph.D. degree. His research interests include role-based management frameworks for network and systems management, role-based access control, management policies and computer supported collaboration tools for organizational structures. He received his diploma (Ingénieur, M.S.) in computer science from the Ecole Nationale Supérieure d'Informatique et de Mathématiques Appliquées de Grenoble (ENSIMAG), Grenoble, France.

Morris Sloman has a Ph.D. in Computer Science from the University of Essex in the U.K. He is a Professor in the Department of Computing at Imperial College, London. His research interests include architecture and languages for heterogeneous distributed systems, distributed systems management and security. He was technical director of the ESPRIT funded SysMan collaborative project which provided domain and policy based tools for managing large inter-organizational distributed systems. He has recently edited a book on Network and Distributed Systems Management and is co-editor of the *Distributed Systems Engineering Journal* which is jointly published by Institution of Electrical Engineers, British Computer Society Institute of Physics.